# Lecture 8

Hierarchical modeling using display lists

Implementing the Interactivity

# Modeling and interactivity

- CAD applications are graphics applications that allow model building interactively (buildings, circuits, etc.)
  - How this interactivity is implemented on the model data level?
    - The model itself is changed during the program execution by addition, removal or modification
    - The model can be saved to file and restored to memory from file
  - This is implemented using a variety of data structures that are designed to support efficient interaction. The model is some data inside these structure. In this way we can imagine saving and modifying a model.

# Hierarchical modeling

- In Hierarchical modeling, an object model can incorporate relationship between parts (inner or constructing objects)
- The face
  - Two eyes (two copies of the eye object)
  - Two ears (two copies of the ear object)
  - A nose (one copy of the nose object)
  - Mouth (one copy of the mouth object)
- The face hierarchical model: specification of the outline, translate to the right eye position then put a copy of the eye object, translate to the left eye position then put a copy from the eye object,…(a set of primitives, relationships and objects)
- The face non-hierarchical model: draw all the face using primitives

# Hierarchical modeling

- In hierarchical modeling, objects are specified using other objects.
- For example, an object A is specified relative to its center which is usually assumed to be located at the origin (object coordinate system).
- Object A can be then used in another object (say B) specification. To put A in its right place as a part of B, the required transformations are done.
- Other object C may be specified using B (which in turn contains A) and so on.

# Hierarchical modeling and display lists

- Because display list can call other display lists, they are powerful tool in building hierarchical models
  - Each object is specified using a list.
  - Each object can be specified in terms of other objects (List that calls other lists)
  - One hierarchical modeling solution for the face example
    - A list to do an eye
    - A list to do an a ear
    - The face list that draw the outline, does the required transformation for the right eye, call the eye list, does the required transformation for the left eye, call the eye list, do the same for the two ears and finally draw the nose and the mouse

# Advantages of hierarchical modeling

- It motivates modularity is design. This simplifies the design and make its modification more easy and structured
- Any component is constructed once and is used any number of time. In OpenGL this can be implemented using display lists, hence leads to performance enhancement.

# Programming event-driven input

- GLUT recognizes a small set of events that are common to most windows system ( standard window systems such as X-windows or MS Windows recognize much more. To exploit these extra events, you need a window system specific library for GL)
- To consume an event, your program register a callback function for it (usually in the main function)
- At the execution time, when the event occurs, the program will invoke your event callback

# Mouse events

- Passive move: Moving the cursor without pressing any button
- Move: moving the mouse while pressing a button
- Mouse event: occur when one of the buttons changes its state (Up-to-down, or down-to-up)
- The position of the mouse cursor is handed to your call back. They are in window coordinate system (in pixels and the origin is the upper left corner of the window)

# Mouse event: registering and callback

Registering fro the mouse event:
Usually written in the main function before the MainLoop function is called

```
glutMouseFunc(myMouse);
```

Write the mouse event handler (call back function) any where in the program, Your function will be called for any mouse event. You can treat each mouse event differently by checking the button state)

```
void myMouse(int button, int state, int x, int y)
{
     if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
               exit(0);
}
```

# Motion event

Registering fro the motion event:
Usually written in the main function before the MainLoop function is called

```
glutMotionFunc(drawSquare);
```

Write the motion event handler (call back function) any where in the program, Your function will be called for any pointing device movement (active movement, ex mouse with button pressed). You can treat each mouse event differently by checking the button state)

```
void drawSquare(int x, int y)
{
      y=wh-y;

      glBegin(GL_POLYGON);
          glVertex2f(x+size, y+size);
          glVertex2f(x-size, y+size);
          glVertex2f(x-size, y-size);
          glVertex2f(x+size, y-size);
      glEnd();
      glFlush();
}
```

# Window events

- Reshape: Occur when the window size is changed, invokes the display callback
  - Do we draw all the shapes that were in the window before it was resized?( note that reshape event invoke display automatically, so if all the drawing is in the display, it will be redrawn after a reshape)
  - What do we do if the aspect ration of the new window is different from that of the old window
  - Do we change the sizes or attributes of new primitives if the window size is changed

# Mouse and window events example

```
/* globals */
GLsizei wh = 500, ww = 500; /* initial window size */
GLfloat size = 3.0;   /* half side length of square */
```

```
void drawSquare(int x, int y)
{
    y=wh-y;// convert from window coordinates to world coordinates
    // choose a random color
    glColor3ub( (char) rand()%256, (char) rand()%256, (char) rand()%256);
    glBegin(GL_POLYGON);
    glVertex2f(x+size, y+size);
    glVertex2f(x-size, y+size);
    glVertex2f(x-size, y-size);
    glVertex2f(x+size, y-size);
    glEnd();
    glFlush();
}
```

```
void myReshape(GLsizei w, GLsizei h)
{
    /* adjust clipping box */
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, (GLdouble)w, 0.0, (GLdouble)h, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    /* adjust viewport and clear */
    glViewport(0,0,w,h);
    glClearColor (1.0, 1.0, 1.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
    /* set global size for use by drawing routine */
    ww = w;
    wh = h;
}
```

```
void myMouse(int btn, int state, int x, int y)
{
    if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN) drawSquare(x,y);
    if(btn==GLUT_RIGHT_BUTTON && state==GLUT_DOWN)   exit(0);
}
void display()
{

}
void myReshape(GLsizei w, GLsizei h)
```

```
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Mouse event");
    glutReshapeFunc(myReshape);
    glutMouseFunc(myMouse);
    // make the program draw rectangle with the mouse move
    //glutMotionFunc(drawSquare);
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
}
```

## Keyboard events

Example of keyboard callback

```
void myKey(unsigned char key, int x, int y)
{
    if(key=='q' || key == 'Q') exit( );
}
```

- Keyboard events are sensed by GLUT when the mouse is in the drawing window and the keyboard is used
  - The glutKeyboardFunc function register a callback for key pressing (they key character and mouse position are the data)
  - The glutKeyboardUpFunc function registers a callback for key releasing (they key character and mouse position are the data)
  - The glutGetModifiers function get the modifiers keys(control, alt , shift)

Using more than on window

```
id=glutCreateWindow("second window");

glutSetWindow(id);
```

## Other tricks for interaction

You can change the call back function at any time anywhere in the program or even remove a callback by setting it to NULL
Use the idle callback to perform tasks when there is no events

Calling glutPostRedisplay ensures that the display will de drawn only once each time the program goes through the event loop

```
glutPostRedisplay();
```